Karlsruhe Reports in Informatics 2010,11
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Dynamic Frames in Java Dynamic Logic

Formalisation and Proofs

Peter H. Schmitt, Mattias Ulbrich, Benjamin Weiß

2010

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Fakultät für **Informatik**

Dynamic Frames in Java Dynamic Logic
Formalisation and Proofs

Peter H. Schmitt, Mattias Ulbrich, and Benjamin Weiß

Karlsruhe Institute of Technology
Institute for Theoretical Computer Science
D-76128 Karlsruhe, Germany
{pschmitt,mulbrich,bweiss}@ira.uka.de

Abstract. This report is a companion to the paper *Dynamic Frames in Java Dynamic Logic* [2]. It contains complementary formal definitions and proofs.

1 Formalisation

1.1 Syntax

Definition 1 (Signatures). *A* signature Σ *is a tuple*

$$\Sigma = (\mathcal{T}, \sqsubseteq, \mathcal{V}, \mathcal{PV}, \mathcal{F}, \mathcal{P}, \alpha, Prg)$$

where $\mathcal{T}$ *is a finite set of* types*; where* $\sqsubseteq$ *is a partial order on* $\mathcal{T}$ *called the* subtype relation*; where* $\mathcal{V}$ *is a set of* (logical) variables*; where* $\mathcal{PV}$ *is a set of* program variables*; where* $\mathcal{F}$ *is a set of* function symbols*; where* $\mathcal{P}$ *is a set of* predicate symbols*; where* α *is a* static typing function *such that* $\alpha(v) \in \mathcal{T}$ *for all* $v \in \mathcal{V} \cup \mathcal{PV}$*,* $\alpha(f) \in \mathcal{T}^* \times \mathcal{T}$ *for all* $f \in \mathcal{F}$*, and* $\alpha(p) \in \mathcal{T}^*$ *for all* $p \in \mathcal{P}$*; and where Prg is some Java program, i.e., a set of Java classes and interfaces.*

We use the notation $v : A$ *for* $\alpha(v) = A$*, the notation* $f : A_1, \ldots, A_n \to A$ *for* $\alpha(f) = ((A_1, \ldots, A_n), A)$*, and the notation* $p : A_1, \ldots, A_n$ *for* $\alpha(p) = (A_1, \ldots, A_n)$*.*

We require that the following types, program variables, function and predicate symbols are present in every signature:

- $Any, Boolean, Int, Null, LocSet, Field, Heap \in \mathcal{T}$
- *all reference types of Prg also appear as types in* $\mathcal{T}$*; in particular,* $Object \in \mathcal{T}$
- *all local variables* `a` *of Prg with Java type* T *also appear as program variables* $\mathtt{a} : A \in \mathcal{PV}$*, where* $A = T$ *if* T *is a reference type,* $A = Boolean$ *if* $T =$ `boolean`*, and* $A = Int$ *if* $T =$ `int` *(in this paper we do not consider other primitive types, and we ignore integer overflows)*
- $\mathtt{heap} : Heap \in \mathcal{PV}$
- $cast_A : Any \to A \in \mathcal{F}$ *(for every type* $A \in \mathcal{T}$*)*
- $TRUE, FALSE : Boolean \in \mathcal{F}$
- $select_A : Heap, Object, Field \to A \in \mathcal{F}$ *(for every type* $A \in \mathcal{T}$*)*

- $store : Heap, Object, Field, Any \to Heap \in \mathcal{F}$
- $anon : Heap, LocSet, Heap \to Heap \in \mathcal{F}$
- $\texttt{null} : Null \in \mathcal{F}$
- *all Java fields* f *of* Prg *also appear as constant symbols* $f : Field \in \mathcal{F}$
- $arr : Int \to Field \in \mathcal{F}$, $created : Field \in \mathcal{F}$
- $allLocs : LocSet \in \mathcal{F}$, $allFields : Object \to LocSet \in \mathcal{F}$, $freshLocs : Heap \to LocSet \in \mathcal{F}$
- $\dot{\emptyset} : LocSet \in \mathcal{F}$, $singleton : Object, Field \to LocSet \in \mathcal{F}$
- $\dot{\cup}, \dot{\cap}, \dot{\setminus} : LocSet, LocSet \to LocSet \in \mathcal{F}$
- $exactInstance_A : Any \in \mathcal{P}$ *(for every type* $A \in \mathcal{T}$*)*
- $wellFormed : Heap \in \mathcal{P}$
- $\doteq : Any, Any \in \mathcal{P}$
- $\dot{\in} : Object, Field, LocSet \in \mathcal{P}$, $\dot{\subseteq}$, $disjoint : LocSet, LocSet \in \mathcal{P}$

We also require that $Boolean, Int, Object, LocSet \sqsubseteq Any$*; that for all* $C \in \mathcal{T}$ *with* $C \sqsubseteq Object$ *we have* $Null \sqsubseteq C$*; that for all types* A, A' *of* Prg *we have* $A' \sqsubseteq A$ *if and only if* A' *is a subtype of* A *in* Prg*; that the types explicitly mentioned in this definition are otherwise unrelated to each other wrt.* $\sqsubseteq$*; and that the types Boolean, Int, Null, LocSet, Field and Heap do not have subtypes except themselves. Finally, we demand that* $\mathcal{V}$*,* $\mathcal{PV}$*,* $\mathcal{F}$ *and* $\mathcal{P}$ *each contain an infinite number of symbols of every typing.*

For illustration, the type hierarchy is visualised in Fig. 1. In the following, we assume a fixed signature $\Sigma = (\mathcal{T}, \sqsubseteq, \mathcal{V}, \mathcal{PV}, \mathcal{F}, \mathcal{P}, \alpha, Prg)$.

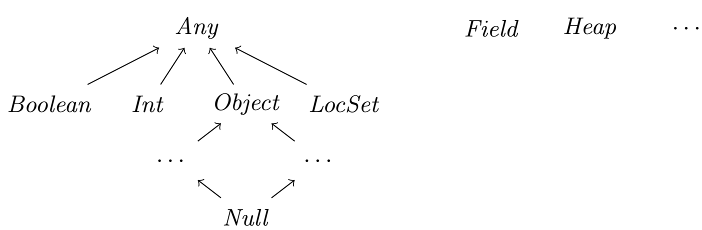

Fig. 1. Type hierarchy

Definition 2 (Syntax). *The sets* Trm^A_Σ *of* terms of type A, Fma_Σ *of* formulas *and* Upd_Σ *of* updates *are defined by the following grammar:*

$$Trm^A_\Sigma ::= x \mid \mathtt{a} \mid f(Trm^{B'_1}_\Sigma, \ldots, Trm^{B'_n}_\Sigma) \mid if(Fma_\Sigma)then(Trm^A_\Sigma)else(Trm^A_\Sigma) \mid \{Upd_\Sigma\}\,Trm^A_\Sigma$$

$$Fma_\Sigma ::= true \mid false \mid p(Trm^{B'_1}_\Sigma, \ldots, Trm^{B'_n}_\Sigma) \mid \neg Fma_\Sigma \mid Fma_\Sigma \wedge Fma_\Sigma \mid Fma_\Sigma \vee Fma_\Sigma \mid Fma_\Sigma \to Fma_\Sigma \mid Fma_\Sigma \leftrightarrow Fma_\Sigma \mid$$

$$\forall A\, x; Fma_\Sigma \;\big|\; \exists A\, x; Fma_\Sigma \;\big|\; [\mathtt{p}]Fma_\Sigma \;\big|\; \langle\mathtt{p}\rangle Fma_\Sigma \;\big|\; \{Upd_\Sigma\}Fma_\Sigma$$

$$Upd_\Sigma ::= \mathtt{a} := Trm_\Sigma^{A'} \;\big|\; Upd_\Sigma \parallel Upd_\Sigma \;\big|\; \{Upd_\Sigma\}Upd_\Sigma$$

for any variable $x : A \in \mathcal{V}$*, any program variable* $\mathtt{a} : A \in \mathcal{PV}$*, any function symbol* $f : B_1, \ldots, B_n \to A \in \mathcal{F}$ *and any predicate symbol* $p : B_1, \ldots, B_n$ *where* $B'_1 \sqsubseteq B_1$*,* $\ldots$*,* $B'_n \sqsubseteq B_n$*, any executable Java fragment* $\mathtt{p}$*, and any type* $A' \in \mathcal{T}$ *with* $A' \sqsubseteq A$*.*

A sequent *is a syntactical construct* $\Gamma \Rightarrow \Delta$*, where* $\Gamma, \Delta \in 2^{Fma_\Sigma}$ *are finite sets of formulas.*

We use infix notation for the binary symbols $\dot\cup$, $\dot\cap$, $\dot\setminus$, $\doteq$, and $\dot\subseteq$. Furthermore, we use the notation $(A)t$ for $cast_A(t)$, the notation $o.f$ for $select_A(\mathtt{heap}, o, f)$ where $f : Field \in \mathcal{F}$ is a Java field, the notation $a[i]$ for $select_A(\mathtt{heap}, a, arr(i))$, the notation $o.*$ for $allFields(o)$, the notation $\{(o, f)\}$ for $singleton(o, f)$, the notation $t_1 \not\doteq t_2$ for $\neg(t_1 \doteq t_2)$, the notation $(o, f) \mathrel{\dot\in} s$ for $\dot\in(o, f, s)$, and the notation $(o, f) \mathrel{\dot\notin} s$ for $\neg(o, f) \mathrel{\dot\in}(s)$.

1.2 Semantics

Definition 3 (Kripke structures). *A* Kripke structure $\mathcal{K}$ *for a signature* Σ *is a tuple*

$$\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$$

where $\mathcal{D}$ *is a set of semantical* values *called the* domain*; where* δ *is a* dynamic typing function $\delta : \mathcal{D} \to \mathcal{T}$*; where (using the definition* $\mathcal{D}^A = \{d \in \mathcal{D} \mid \delta(d) \sqsubseteq A\}$*)* I *is an* interpretation function *that maps every function symbol* $f : A_1, \ldots, A_n \to A \in \mathcal{F}$ *to a function* $I(f) : \mathcal{D}^{A_1}, \ldots, \mathcal{D}^{A_n} \to \mathcal{D}^A$ *and every predicate symbol* $p : A_1, \ldots, A_n \in \mathcal{P}$ *to a relation* $I(p) \subseteq \mathcal{D}^{A_1} \times \cdots \times \mathcal{D}^{A_n}$*; where* $\mathcal{S}$ *is the set of all* states*, which are functions* $s \in \mathcal{S}$ *mapping every program variable* $\mathtt{a} : A \in \mathcal{PV}$ *to a value* $s(a) \in \mathcal{D}^A$*; and where* ρ *is a function associating with every executable Java fragment* $\mathtt{p}$ *in the context of Prg a* transition relation $\rho(\mathtt{p}) \subseteq \mathcal{S}^2$ *such that* $(s_1, s_2) \in \rho(\mathtt{p})$ *iff* $\mathtt{p}$*, when started in* s_1*, terminates normally in* s_2 *(according to the Java semantics [1]). We consider Java programs to be deterministic, so for all program fragments* $\mathtt{p}$ *and all* $s_1 \in \mathcal{S}$*, there is at most one* s_2 *such that* $(s_1, s_2) \in \rho(\mathtt{p})$*.*

We require that every Kripke structure satisfies the following:

- $\mathcal{D}^{Boolean} = \{tt, ff\}$, $\mathcal{D}^{Int} = \mathbb{Z}$, $\mathcal{D}^{Null} = \{I(\mathtt{null})\}$, $\mathcal{D}^{LocSet} = 2^{\mathcal{D}^{Object} \times \mathcal{D}^{Field}}$, $\mathcal{D}^{Heap} = \mathcal{D}^{Object} \times \mathcal{D}^{Field} \to \mathcal{D}^{Any}$
- $\delta(d) \neq T$ *for all* $d \in \mathcal{D}$*, if* $T \in \mathcal{T}$ *represents an interface or an abstract class*
- $\{d \in \mathcal{D} \mid \delta(d) = T\}$ *is infinite for all* $T \sqsubseteq Object$*,* $T \neq Null$ *not representing an interface or an abstract class*
- $I(cast_A)(d) = d$ *for all* $d \in \mathcal{D}^A$
- $I(TRUE) = tt$*,* $I(FALSE) = ff$
- $I(select_A)(h, o, f) = I(cast_A)(h(o, f))$ *for all* $h \in \mathcal{D}^{Heap}$*,* $o \in \mathcal{D}^{Object}$*,* $f \in \mathcal{D}^{Field}$

– $I(store)(h, o, f, d)(o', f') = \begin{cases} d & \text{if } o = o' \text{ and } f = f' \\ h(o', f') & \text{otherwise} \end{cases}$
  for all $h \in \mathcal{D}^{Heap}$, $o, o' \in \mathcal{D}^{Object}$, $f, f' \in \mathcal{D}^{Field}$, $d \in \mathcal{D}^{Any}$
– $I(anon)(h, s, h')(o, f) = \begin{cases} h'(o, f) & \text{if } \big((o, f) \in s \text{ and } f \neq I(created)\big) \\ & \text{or } (o, f) \in I(freshLocs)(h) \\ h(o, f) & \text{otherwise} \end{cases}$
  for all $h, h' \in \mathcal{D}^{Heap}$, $s \in \mathcal{D}^{LocSet}$, $o \in \mathcal{D}^{Object}$, $f \in \mathcal{D}^{Field}$
– *let UniqueFunctions* $\subseteq \mathcal{F}$ *be the set consisting of the constant symbols representing Java fields, of arr and of created; then we require that for all* $f, g \in$ *UniqueFunctions the function* $I(f)$ *is injective, and that the ranges of the functions* $I(f)$ *and* $I(g)$ *are disjoint.*
– $I(allLocs) = \mathcal{D}^{Object} \times \mathcal{D}^{Field}$, $I(allFields)(o) = \{(o, f) \mid f \in \mathcal{D}^{Field}\}$,
  $I(freshLocs)(h) = \{(o, f) \in I(allLocs) \mid o \neq I(\mathtt{null}), h(o, I(created)) = f\!f\}$
– $I(\dot{\emptyset}) = \emptyset$, $I(singleton)(o, f) = \{(o, f)\}$, $I(\dot{\cup}) = \cup$, $I(\dot{\cap}) = \cap$, $I(\dot{\setminus}) = \setminus$
– $I(exactInstance_A) = \{d \in \mathcal{D} \mid \delta(d) = A\}$
– $I(wellFormed) = \{h \in \mathcal{D}^{Heap} \mid$ *for all* $o \in \mathcal{D}^{Object}, f \in \mathcal{D}^{Field}$:
  if $h(o, f) \in \mathcal{D}^{Object}$, *then* $h(o, f) = I(\mathtt{null})$
  or $h(h(o, f), I(created)) = tt\}$
– $I(\doteq) = \{(d, d) \in \mathcal{D}^2\}$
– $I(\dot{\in}) = \{(o, f, s) \in \mathcal{D}^{Object} \times \mathcal{D}^{Field} \times \mathcal{D}^{LocSet} \mid (o, f) \in s\}$, $I(\dot{\subseteq}) = \{(s_1, s_2) \in (\mathcal{D}^{LocSet})^2 \mid s_1 \subseteq s_2\}$, $I(disjoint) = \{(s_1, s_2) \in (\mathcal{D}^{LocSet})^2 \mid s_1 \cap s_2 = \emptyset\}$

Definition 4 (Semantics). *Given a Kripke structure* $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$, *a state* $s \in \mathcal{S}$ *and a variable assignment* $\beta : \mathcal{V} \to \mathcal{D}$ *(where for every* $x : A \in \mathcal{V}$ *we have* $\beta(x) \in \mathcal{D}^A$*), we evaluate terms* $t \in Trm_\Sigma^A$ *to a value* $val_{\mathcal{K},s,\beta}(t) \in \mathcal{D}^A$*, formulas* $\varphi \in Fma_\Sigma$ *to a truth value* $val_{\mathcal{K},s,\beta}(\varphi) \in \{tt, f\!f\}$*, and updates* $u \in Upd_\Sigma$ *to a state transformer* $val_{\mathcal{K},s,\beta}(u) : \mathcal{S} \to \mathcal{S}$ *as defined below.*

$$\begin{aligned}
val_{\mathcal{K},s,\beta}(x) &= \beta(x) \\
val_{\mathcal{K},s,\beta}(\mathtt{a}) &= s(\mathtt{a}) \\
val_{\mathcal{K},s,\beta}(f(t_1, \ldots, t_n)) &= I(f)(val_{\mathcal{K},s,\beta}(t_1), \ldots, val_{\mathcal{K},s,\beta}(t_n)) \\
val_{\mathcal{K},s,\beta}(if(\varphi)then(t_1)else(t_2)) &= \begin{cases} val_{\mathcal{K},s,\beta}(t_1) \text{ if } val_{\mathcal{K},s,\beta}(\varphi) = tt \\ val_{\mathcal{K},s,\beta}(t_2) \text{ otherwise} \end{cases} \\
val_{\mathcal{K},s,\beta}(\{u\}t) &= val_{\mathcal{K},s',\beta}(t), \text{ where } s' = val_{\mathcal{K},s,\beta}(u)(s) \\
val_{\mathcal{K},s,\beta}(true) &= tt \\
val_{\mathcal{K},s,\beta}(false) &= f\!f \\
val_{\mathcal{K},s,\beta}(p(t_1, \ldots, t_n)) &= tt \text{ iff } (val_{\mathcal{K},s,\beta}(t_1), \ldots, val_{\mathcal{K},s,\beta}(t_n)) \in I(p) \\
val_{\mathcal{K},s,\beta}(\neg\varphi) &= tt \text{ iff } val_{\mathcal{K},s,\beta}(\varphi) = f\!f \\
val_{\mathcal{K},s,\beta}(\varphi_1 \wedge \varphi_2) &= tt \text{ iff } f\!f \notin \{val_{\mathcal{K},s,\beta}(\varphi_1), val_{\mathcal{K},s,\beta}(\varphi_2)\} \\
val_{\mathcal{K},s,\beta}(\varphi_1 \vee \varphi_2) &= tt \text{ iff } tt \in \{val_{\mathcal{K},s,\beta}(\varphi_1), val_{\mathcal{K},s,\beta}(\varphi_2)\} \\
val_{\mathcal{K},s,\beta}(\varphi_1 \to \varphi_2) &= val_{\mathcal{K},s,\beta}(\neg\varphi_1 \vee \varphi_2) \\
val_{\mathcal{K},s,\beta}(\varphi_1 \leftrightarrow \varphi_2) &= val_{\mathcal{K},s,\beta}(\varphi_1 \to \varphi_2 \wedge \varphi_2 \to \varphi_1)
\end{aligned}$$

$$
\begin{aligned}
val_{\mathcal{K},s,\beta}(\forall A\, x; \varphi) &= tt \textit{ iff } ff \notin \{val_{\mathcal{K},s,\beta_x^d}(\varphi) \mid d \in \mathcal{D}^A\}\\
val_{\mathcal{K},s,\beta}(\exists A\, x; \varphi) &= tt \textit{ iff } tt \in \{val_{\mathcal{K},s,\beta_x^d}(\varphi) \mid d \in \mathcal{D}^A\}\\
val_{\mathcal{K},s,\beta}([\mathtt{p}]\varphi) &= tt \textit{ iff } ff \notin \{val_{\mathcal{K},s',\beta}(\varphi) \mid (s,s') \in \rho(\mathtt{p})\}\\
val_{\mathcal{K},s,\beta}(\langle\mathtt{p}\rangle\varphi) &= tt \textit{ iff } tt \in \{val_{\mathcal{K},s',\beta}(\varphi) \mid (s,s') \in \rho(\mathtt{p})\}\\
val_{\mathcal{K},s,\beta}(\{u\}\varphi) &= val_{\mathcal{K},s',\beta}(\varphi), \textit{ where } s' = val_{\mathcal{K},s,\beta}(u)(s)\\
val_{\mathcal{K},s,\beta}(\mathtt{a} := t)(s')(\mathtt{b}) &= \begin{cases} val_{\mathcal{K},s,\beta}(t) & \textit{if } \mathtt{b} = \mathtt{a}\\ s'(\mathtt{b}) & \textit{otherwise}\end{cases}\\
&\quad \textit{for all } s' \in \mathcal{S},\ \mathtt{b} \in \mathcal{PV}\\
val_{\mathcal{K},s,\beta}(u_1 \parallel u_2)(s') &= val_{\mathcal{K},s,\beta}(u_2)(val_{\mathcal{K},s,\beta}(u_1)(s')) \textit{ for all } s' \in \mathcal{S}\\
val_{\mathcal{K},s,\beta}(\{u_1\}u_2) &= val_{\mathcal{K},s',\beta}(u_2), \textit{ where } s' = val_{\mathcal{K},s,\beta}(u_1)(s)
\end{aligned}
$$

We sometimes write $(\mathcal{K}, s, \beta) \models \varphi$ *instead of* $val_{\mathcal{K},s,\beta}(\varphi) = tt$. *A formula* $\varphi \in Fma_\Sigma$ *is called* logically valid, *in symbols* $\models \varphi$, *iff* $(\mathcal{K}, s, \beta) \models \varphi$ *for all Kripke structures* $\mathcal{K}$, *all states* $s \in \mathcal{S}$, *and all variable assignments* β.

The semantics of a sequent $\Gamma \Rightarrow \Delta$ *is the same as that of a formula* $\bigwedge \Gamma \to \bigvee \Delta$, *where* $\bigvee\{\varphi_1, \ldots, \varphi_n\} = \varphi_1 \vee \cdots \vee \varphi_n$, *and* $\bigwedge\{\varphi_1, \ldots, \varphi_n\} = \varphi_1 \wedge \cdots \wedge \varphi_n$.

1.3 Observations

The propositions below are used as assumptions in the proofs in Sect. 2. We do not prove them, but consider them obvious.

Proposition 1 (Non-occurring program variables). *For all Kripke structures* $\mathcal{K}$, *all states* $s, s' \in \mathcal{S}$, *all variable assignments* β, *and all* $t \in Trm_\Sigma \cup Fma_\Sigma \cup Upd_\Sigma$: *if for all program variables* $\mathtt{a} \in \mathcal{PV}$ *that syntactically occur in* t *we have* $s(\mathtt{a}) = s'(\mathtt{a})$, *then we also have* $val_{\mathcal{K},s,\beta}(t) = val_{\mathcal{K},s',\beta}(t)$.

Note that a program variable `b` not occurring in t *can* play a role in evaluating t, namely if t contains a program which calls a method that in turn manipulates `b`. Still, in a Java program a called method can never read the value of a local variable `b` before assigning to `b`; thus, the initial value of `b` as defined by s or s' does not matter. We consider `heap` $\in \mathcal{PV}$ to implicitly occur in field access expressions `o.f`, in array access expressions `a[i]`, and in method calls `o.m(...)`.

Proposition 2 (Non-occurring function and predicate symbols). *For all Kripke structures* $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$ *and* $\mathcal{K}' = (\mathcal{D}, \delta, I', \mathcal{S}, \rho)$ *differing only in the interpretation functions* I *vs.* I', *all states* $s \in \mathcal{S}$, *all variable assignments* β, *and all* $t \in Trm_\Sigma \cup Fma_\Sigma \cup Upd_\Sigma$: *if for all function and predicate symbols* $f \in \mathcal{F} \cup \mathcal{P}$ *that syntactically occur in* t *we have* $I(f) = I'(f)$, *then we also have* $val_{\mathcal{K},s,\beta}(t) = val_{\mathcal{K}',s,\beta}(t)$.

Proposition 3 (Overwritten program variables). *For all Kripke structures* $\mathcal{K}$, *all states* $s, s' \in \mathcal{S}$, *all variable assignments* β, *all updates* $(\mathtt{a} := t') \in Upd_\Sigma$

where `a` *does not occur in* t'*, all* $t \in Trm_\Sigma \cup Fma_\Sigma \cup Upd_\Sigma$*, all* $\varphi \in Fma_\Sigma$*, and all program fragments* `p`*: if for all program variables* $\mathtt{b} \in \mathcal{PV} \setminus \{\mathtt{a}\}$ *which occur in* t *or* φ *we have* $s(\mathtt{b}) = s'(\mathtt{b})$*, then we also have:*

$$val_{\mathcal{K},s,\beta}(\{\mathtt{a} := t'\}t) = val_{\mathcal{K},s',\beta}(\{\mathtt{a} := t'\}t)$$
$$val_{\mathcal{K},s,\beta}([\mathtt{a\ =\ } t'\mathtt{;\ p}]\varphi) = val_{\mathcal{K},s',\beta}([\mathtt{a\ =\ } t'\mathtt{;\ p}]\varphi)$$
$$val_{\mathcal{K},s,\beta}(\langle\mathtt{a\ =\ } t'\mathtt{;\ p}\rangle\varphi) = val_{\mathcal{K},s',\beta}(\langle\mathtt{a\ =\ } t'\mathtt{;\ p}\rangle\varphi)$$

Prop. 3 holds because the initial value of the program variable `a` is overwritten by the preceding update or assignment, and thus cannot influence the evaluation of t or φ, respectively.

Proposition 4 (Method calls). *Let* `p` *be a method call statement* ($\mathtt{res = this.m(p_1, \ldots, p_n);}$)*, let* $\mathtt{hPre}: Heap \in \mathcal{PV}$*, let* $reachableState \in Fma_\Sigma$ *be as in Def. 3 of [2], let* $reachableState' \in Fma_\Sigma$ *be as in Def. 4, and let* $noDeallocs \in Fma_\Sigma$ *be as in Def. 7. Then the following holds:*

$$\models reachableState \rightarrow \{\mathtt{hPre} := \mathtt{heap}\}[\mathtt{p}](reachableState' \wedge noDeallocs)$$

Prop. 4 is guaranteed by the semantics of Java.

2 Proofs

2.1 Preparation

Lemma 1 (Relation between *frame* **and** *anon***).** *Let* $mod \in Trm_\Sigma^{LocSet}$*,* $\mathtt{hPre}: Heap \in \mathcal{PV}$*,* $frame \in Fma_\Sigma$ *be as in Def. 3 of [2],* $noDeallocs \in Fma_\Sigma$ *be as in Def. 7, and let* $frame' \in Fma_\Sigma$ *be the formula*

$$\mathtt{heap} \doteq anon(\mathtt{hPre}, \{\mathtt{heap} := \mathtt{hPre}\}mod, \mathtt{heap}).$$

Then the following holds:

$$\models (frame \wedge noDeallocs) \leftrightarrow frame'$$

Proof. Let $\mathcal{K}$ be a Kripke structure, $s \in \mathcal{S}$ be a state, β be a variable assignment, $h = s(\mathtt{heap})$, $h' = val_{\mathcal{K},s,\beta}(anon(\mathtt{hPre}, \{\mathtt{heap} := \mathtt{hPre}\}mod, \mathtt{heap})$, $s^{pre} = val_{\mathcal{K},s,\beta}(\mathtt{heap} := \mathtt{hPre})(s)$, $h^{pre} = s^{pre}(\mathtt{heap})$, $m^{pre} = val_{\mathcal{K},s^{pre},\beta}(mod)$, $fl = I(freshLocs)(h)$, and $fl^{pre} = I(freshLocs)(h^{pre})$. Note that $h^{pre} = s(\mathtt{hPre})$. By definition of $I(anon)$, we know that the following holds for all $o \in \mathcal{D}^{Object}$, $f \in \mathcal{D}^{Field}$:

$$h'(o,f) = \begin{cases} h(o,f) & \text{if } \big((o,f) \in m^{pre} \text{ and } f \neq I(created)\big) \\ & \text{or } (o,f) \in fl^{pre} \\ h^{pre}(o,f) & \text{otherwise} \end{cases} \qquad (1)$$

We first show that $(\mathcal{K}, s, \beta) \models frame \wedge noDeallocs$ implies that $(\mathcal{K}, s, \beta) \models frame'$, and then the other way round.

1. Let $o \in \mathcal{D}^{Object}$, $f \in \mathcal{D}^{Field}$. Using the definitions of *frame*, *noDeallocs* and *frame'*, we assume

$$(o, f) \in m^{pre} \cup \mathit{fl}^{pre} \quad \text{or} \quad h(o, f) = h^{pre}(o, f) \tag{2}$$
$$\text{if } (o, f) \in \mathit{fl}, \text{ then } (o, f) \in \mathit{fl}^{pre} \tag{3}$$
$$h(I(\texttt{null}), I(\mathit{created})) = h^{pre}(I(\texttt{null}), I(\mathit{created})) \tag{4}$$

and aim to show
$$h'(o, f) = h(o, f). \tag{5}$$

From (2) we get that one of the following three cases must apply:

- $(o, f) \in m^{pre}$. If $f \neq I(\mathit{created})$ or $(o, f) \in \mathit{fl}^{pre}$, then (5) immediately follows from (1). We thus assume

$$f = I(\mathit{created}) \tag{6}$$
$$(o, f) \notin \mathit{fl}^{pre}. \tag{7}$$

  Now, (1) yields
$$h'(o, f) = h^{pre}(o, f). \tag{8}$$

  If $o = I(\texttt{null})$, then we get from (4) that $h(o, f) = h^{pre}(o, f)$, which together with (8) immediately yields (5). Thus we assume

$$o \neq I(\texttt{null}). \tag{9}$$

  From (3) and (7) we get that

$$(o, f) \notin \mathit{fl}.$$

  This, (9), and the definition of $I(\mathit{freshLocs})$ imply $h(o, I(\mathit{created})) = \mathit{tt}$. Analogously, (7) and (9) imply $h^{pre}(o, I(\mathit{created})) = \mathit{tt}$. Together, we have $h(o, I(\mathit{created})) = h^{pre}(o, I(\mathit{created}))$, which because of (6) can be written as $h(o, f) = h^{pre}(o, f)$. We combine this with (8) to get (5).
- $(o, f) \in \mathit{fl}^{pre}$. Then (1) immediately yields (5).
- $h(o, f) = h^{pre}(o, f)$. If $(o, f) \in m^{pre}$ or $(o, f) \in \mathit{fl}^{pre}$, then the proof proceeds as for the respective case above. Otherwise, (1) guarantees that $h'(o, f) = h^{pre}(o, f)$, and thus we have (5).

2. Let $o \in \mathcal{D}^{Object}$, $f \in \mathcal{D}^{Field}$. We assume (5), and show first (2), then (3), and finally (4).
   (a) If $(o, f) \in m^{pre}$ or $(o, f) \in \mathit{fl}^{pre}$, then (2) holds trivially. Otherwise, (5) and (1) imply $h(o, f) = h^{pre}(o, f)$, which also implies (2).
   (b) We prove (3) by contradiction: we assume that $(o, f) \in \mathit{fl} \setminus \mathit{fl}^{pre}$. By definition of $I(\mathit{freshLocs})$, this means that $o \neq I(\texttt{null})$, that $h(o, I(\mathit{created})) = \mathit{ff}$, and that $h^{pre}(o, I(\mathit{created})) = \mathit{tt}$. From (5) and (1) we get that $h(o, I(\mathit{created})) = h^{pre}(o, I(\mathit{created}))$. Together, we have $\mathit{ff} = \mathit{tt}$.
   (c) The definition of $I(\mathit{freshLocs})$ tells us that $(I(\texttt{null}), I(\mathit{created})) \notin \mathit{fl}^{pre}$. Thus, (5) and (1) immediately guarantee (4). □

2.2 Method Contracts

Theorem 1 (Soundness of useMethodContract). *Let* $\Gamma, \Delta \in 2^{Fma_\Sigma}$*,* $u \in Upd_\Sigma$*,* $[\![\cdot]\!] \in \{[\cdot], \langle\cdot\rangle\}$*,* $\mathtt{r} \in \mathcal{PV}$*,* $\mathtt{o} \in Trm_\Sigma$*, the method* $\mathtt{m}$*,* $\mathtt{p}'_1, \ldots, \mathtt{p}'_n \in Trm_\Sigma$*,* $\varphi \in Fma_\Sigma$*,* $A \in \mathcal{T}$*,* $mct = (\mathtt{m}, \mathtt{this}, (\mathtt{p}_1, \ldots, \mathtt{p}_n), \mathtt{res}, \mathtt{hPre}, pre, post, mod, \tau)$*,* $reachableState, reachableState' \in Fma_\Sigma$*,* $v, w \in Upd_\Sigma$*, and* $h, r' \in \mathcal{F}$ *all be as in Def. 4 of [2]. If*

$$\models \Gamma \Rightarrow \{u\}\{w\}(pre \wedge reachableState), \Delta \tag{10}$$

$$\models \Gamma \Rightarrow \{u\}\{w\}\{\mathtt{hPre} := \mathtt{heap}\}\{v\}(post \wedge reachableState' \rightarrow [\![\ldots]\!]\varphi), \Delta \tag{11}$$

and if for all types $B \sqsubseteq A$ *we have*

$$\models CorrectMethodContract(mct, B), \tag{12}$$

then the following holds:

$$\models \Gamma \Rightarrow \{u\}[\![\mathtt{r\ =\ o.m(p'_1, \ldots, p'_n);\ \ldots}]\!]\varphi, \Delta.$$

Proof. Let (10), (11) and (12) hold. Let furthermore $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$ be a Kripke structure, $s \in \mathcal{S}$, and β be a variable assignment. Our goal is to show

$$(\mathcal{K}, s, \beta) \models\ \Gamma \Rightarrow \{u\}[\![\mathtt{r\ =\ o.m(p'_1, \ldots, p'_n);\ \ldots}]\!]\varphi, \Delta.$$

If there is $\gamma \in \Gamma$ with $val_{\mathcal{K},s,\beta}(\gamma) = ff$ or if there is $\delta \in \Delta$ with $val_{\mathcal{K},s,\beta}(\delta) = tt$, then this is trivially true. We therefore assume that

$$(\mathcal{K}, s, \beta) \models \bigwedge(\Gamma \cup \neg\Delta), \tag{13}$$

and aim to show that $(\mathcal{K}, s, \beta) \models \{u\}[\![\mathtt{r\ =\ o.m(p'_1, \ldots, p'_n);\ \ldots}]\!]\varphi$.

Let $s_1 = val_{\mathcal{K},s,\beta}(u)(s)$. Then our goal is to show

$$(\mathcal{K}, s_1, \beta) \models [\![\mathtt{r\ =\ o.m(p'_1, \ldots, p'_n);\ \ldots}]\!]\varphi.$$

Let $s_2 = val_{\mathcal{K},s_1,\beta}(w)(s_1)$. Because of the definition of w, it holds for all $\mathtt{a} \in \mathcal{PV} \setminus \{\mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n\}$ that $s_1(\mathtt{a}) = s_2(\mathtt{a})$. Since by Def. 4 neither $\mathtt{this}$ nor $\mathtt{p}_1, \ldots, \mathtt{p}_n$ occur in the above formula, Prop. 1 tells us that the interpretation of this formula is the same in s_1 and s_2. It is therefore sufficient if we show

$$(\mathcal{K}, s_2, \beta) \models [\![\mathtt{r\ =\ o.m(p'_1, \ldots, p'_n);\ \ldots}]\!]\varphi.$$

The definition of w and Prop. 1 ensure that $s_2(\mathtt{this}) = val_{\mathcal{K},s_2,\beta}(\mathtt{o})$, and that $s_2(\mathtt{p}_1) = val_{\mathcal{K},s_2,\beta}(\mathtt{p}'_1)$, ..., $s_2(\mathtt{p}_n) = val_{\mathcal{K},s_2,\beta}(\mathtt{p}'_n)$. Thus, we can aim to prove the formula below instead of the formula above:

$$(\mathcal{K}, s_2, \beta) \models [\![\mathtt{r\ =\ this.m(p_1, \ldots, p_n);\ \ldots}]\!]\varphi.$$

Since by Def. 4 the program variable `res` does not occur in the above formula, the Java semantics allows us to instead show

$$(\mathcal{K}, s_2, \beta) \models [\![\mathtt{res\ =\ this.m(p_1, \ldots, p_n);\ r\ =\ res;\ \ldots}]\!]\varphi.$$

Let $s_3 = val_{\mathcal{K},s_2,\beta}(\texttt{hPre} := \texttt{heap})(s_2)$. Since by Def. 4 the program variable `hPre` does not occur in the above formula, by Prop. 1 it is sufficient if we prove

$$(\mathcal{K}, s_3, \beta) \models [\![\texttt{res = this.m(}\mathtt{p}_1, \ldots, \mathtt{p}_n\texttt{); r = res; }\ldots]\!]\varphi. \tag{thm1-goal}$$

We combine (13) with (10) to get

$$(\mathcal{K}, s, \beta) \models \{u\}\{w\}(pre \wedge reachableState),$$

which by definition of s_2 is the same as

$$(\mathcal{K}, s_2, \beta) \models pre \wedge reachableState. \tag{14}$$

Let $C = \delta(s_2(\texttt{this}))$. This means that

$$(\mathcal{K}, s_2, \beta) \models exactInstance_C(\texttt{this}). \tag{15}$$

Since $\alpha(\texttt{this}) = A$, we have $C \sqsubseteq A$ because of well-typedness. Instantiating (12) with C and s_2 yields

$$\begin{aligned}(\mathcal{K}, s_2, \beta) \models\ & pre \wedge reachableState \wedge exactInstance_C(\texttt{this}) \\ & \rightarrow \{\texttt{hPre} := \texttt{heap}\}[\![\texttt{res = this.m(}\mathtt{p}_1, \ldots, \mathtt{p}_n\texttt{);}]\!]'(post \wedge frame)\end{aligned}$$

where $[\![\cdot]\!]'$ is $\langle\cdot\rangle$ if $[\![\cdot]\!]$ is $\langle\cdot\rangle$, and where $[\![\cdot]\!]'$ is either $\langle\cdot\rangle$ or $[\cdot]$ otherwise. Together with (14) and (15), this implies

$$(\mathcal{K}, s_2, \beta) \models \{\texttt{hPre} := \texttt{heap}\}[\![\texttt{res = this.m(}\mathtt{p}_1, \ldots, \mathtt{p}_n\texttt{);}]\!]'(post \wedge frame).$$

With the definition of s_3, this becomes

$$(\mathcal{K}, s_3, \beta) \models [\![\texttt{res = this.m(}\mathtt{p}_1, \ldots, \mathtt{p}_n\texttt{);}]\!]'(post \wedge frame). \tag{16}$$

If there is no $s_4 \in \mathcal{S}$ such that $(s_3, s_4) \in \rho(\texttt{res = this.m(}\mathtt{p}_1, \ldots, \mathtt{p}_n\texttt{);})$ (i.e., if the method call does not terminate when started in s_3), then (16) implies that $[\![\cdot]\!]'$ must be $[\cdot]$, and thus $[\![\cdot]\!]$ also must be $[\cdot]$. Then, (thm1-goal) holds trivially, because there is no final state which would have to satisfy φ.

We can thus find $s_4 \in \mathcal{S}$ such that $(s_3, s_4) \in \rho(\texttt{res = this.m(}\mathtt{p}_1, \ldots, \mathtt{p}_n\texttt{);})$. As our programs are deterministic, s_4 is the only such state. Our proof goal (thm1-goal) now becomes

$$(\mathcal{K}, s_4, \beta) \models [\![\texttt{r = res; }\ldots]\!]\varphi. \tag{thm1-goal'}$$

From (16) and the definition of s_4 we get

$$(\mathcal{K}, s_4, \beta) \models post \wedge frame. \tag{17}$$

Let $noDeallocs \in Fma_\Sigma$ be as in Def. 7. Prop. 4 tells us that

$$\begin{aligned}(\mathcal{K}, s_2, \beta) \models\ & reachableState \\ & \rightarrow \{\texttt{hPre} := \texttt{heap}\}[\texttt{res = this.m(}\mathtt{p}_1, \ldots, \mathtt{p}_n\texttt{;)}] \\ & \qquad (reachableState' \wedge noDeallocs).\end{aligned}$$

Together with (14) and the definition of s_4, this turns into

$$(\mathcal{K}, s_4, \beta) \models \mathit{reachableState}' \wedge \mathit{noDeallocs}. \tag{18}$$

Let $\mathcal{K}' = (\mathcal{D}, \delta, I', \mathcal{S}, \rho)$ be a Kripke structure identical to $\mathcal{K}$, except that $I'(h) = s_4(\texttt{heap})$, and except that $I'(r') = s_4(\texttt{res})$. Since by Def. 4 the symbols h and r' do not occur in Γ nor in Δ, we get from (13) that $(\mathcal{K}', s, \beta) \models \bigwedge(\Gamma \cup \neg\Delta)$. This and (11) imply

$$(\mathcal{K}', s, \beta) \models \{u\}\{w\}\{\texttt{hPre} := \texttt{heap}\}\{v\}(\mathit{post} \wedge \mathit{reachableState}' \rightarrow [\![\ldots]\!]\varphi).$$

As h and r' do not occur in u, in w or in $\texttt{hPre} := \texttt{heap}$, the above and Prop. 2 imply that

$$(\mathcal{K}', s_3, \beta) \models \{v\}(\mathit{post} \wedge \mathit{reachableState}' \rightarrow [\![\ldots]\!]\varphi).$$

Let $s'_4 = \mathit{val}_{\mathcal{K}', s_3, \beta}(v)(s_3)$. Then the above implies

$$(\mathcal{K}', s'_4, \beta) \models \mathit{post} \wedge \mathit{reachableState}' \rightarrow [\![\ldots]\!]\varphi.$$

Since h and r' do not occur in the above formula, by Prop. 2 we get that

$$(\mathcal{K}, s'_4, \beta) \models \mathit{post} \wedge \mathit{reachableState}' \rightarrow [\![\ldots]\!]\varphi. \tag{19}$$

Given the definition of s_4, the semantics of Java tells us that for all $a \in \mathcal{PV} \setminus \{\texttt{heap}, \texttt{res}\}$ we have $s_3(\texttt{a}) = s_4(\texttt{a})$. Similarly, the definition of s'_4 implies that for all $\texttt{a} \in \mathcal{PV} \setminus \{\texttt{heap}, \texttt{r}, \texttt{res}\}$ we have $s_3(\texttt{a}) = s'_4(\texttt{a})$. Together, we have

$$\text{for all } \texttt{a} \in \mathcal{PV} \setminus \{\texttt{heap}, \texttt{r}, \texttt{res}\} : s'_4(\texttt{a}) = s_4(\texttt{a}). \tag{20}$$

The definition of s'_4 also guarantees that

$$s'_4(\texttt{heap}) = \mathit{val}_{\mathcal{K}', s_3, \beta}(\mathit{anon}(\texttt{heap}, \mathit{mod}, h)) \tag{21}$$

$$s'_4(\texttt{r}) = I'(r') = s_4(\texttt{res}) \tag{22}$$

$$s'_4(\texttt{res}) = I'(r') = s_4(\texttt{res}) \tag{23}$$

Using (17) and (18), Lemma 1 tells us that

$$(\mathcal{K}, s_4, \beta) \models \texttt{heap} \doteq \mathit{anon}(\texttt{hPre}, \{\texttt{heap} := \texttt{hPre}\}\mathit{mod}, \texttt{heap}),$$

which we can also express as

$$s_4(\texttt{heap}) = \mathit{val}_{\mathcal{K}, s_4, \beta}(\mathit{anon}(\texttt{hPre}, \{\texttt{heap} := \texttt{hPre}\}\mathit{mod}, \texttt{heap})).$$

Since by Def. 4 the function symbols h and r' do not occur in the above formula, and since $\mathcal{K}'$ is otherwise identical to $\mathcal{K}$, Prop. 2 yields

$$s_4(\texttt{heap}) = \mathit{val}_{\mathcal{K}', s_4, \beta}(\mathit{anon}(\texttt{hPre}, \{\texttt{heap} := \texttt{hPre}\}\mathit{mod}, \texttt{heap})).$$

As we defined $\mathcal{K}'$ such that $I'(h) = s_4(\texttt{heap})$, this implies

$$s_4(\texttt{heap}) = val_{\mathcal{K}',s_4,\beta}(anon(\texttt{hPre}, \{\texttt{heap} := \texttt{hPre}\}mod, h)).$$

Since s_3 and s_4 are identical except for `heap` and `res`, and since `res` does not occur in $\{\texttt{heap} := \texttt{hPre}\}mod$, Prop. 3 tells us that $val_{\mathcal{K},s_4,\beta}(\{\texttt{heap} := \texttt{hPre}\}mod) = val_{\mathcal{K},s_3,\beta}(\{\texttt{heap} := \texttt{hPre}\}mod)$. As `heap` and `res` do not occur in the other arguments of *anon*, we can transform the statement above into

$$s_4(\texttt{heap}) = val_{\mathcal{K}',s_3,\beta}(anon(\texttt{hPre}, \{\texttt{heap} := \texttt{hPre}\}mod, h)).$$

The definition of s_3 implies $s_3(\texttt{heap}) = s_3(\texttt{hPre})$. Thus, the update $\texttt{heap} := \texttt{hPre}$ has no effect in s_3. This allows simplifying the above into

$$s_4(\texttt{heap}) = val_{\mathcal{K}',s_3,\beta}(anon(\texttt{hPre}, mod, h)),$$

and replacing `hPre` with `heap` to get

$$s_4(\texttt{heap}) = val_{\mathcal{K}',s_3,\beta}(anon(\texttt{heap}, mod, h)).$$

This, together with (21), implies that $s_4(\texttt{heap}) = s'_4(\texttt{heap})$. Combining this result with (20) and (23) yields that s_4 and s'_4 differ at most in `r`. Since by Def. 4 the program variable `r` does not occur in *post*, (17) and Prop. 1 imply

$$(\mathcal{K}, s'_4, \beta) \models post. \tag{24}$$

As `r` also does not occur in *reachableState*$'$, we get from (18) that

$$(\mathcal{K}, s'_4, \beta) \models reachableState'.$$

This, (24) and (19) together imply

$$(\mathcal{K}, s'_4, \beta) \models [\![\ldots]\!]\varphi.$$

By (22) and (23), we know that $s'_4(\texttt{res}) = s'_4(\texttt{r})$. Thus, the Java semantics allows us to rewrite the above statement into

$$(\mathcal{K}, s'_4, \beta) \models [\![\texttt{r = res;}\ldots]\!]\varphi.$$

Finally, as s_4 and s'_4 differ at most in `r`, Prop. 3 tells us that

$$(\mathcal{K}, s_4, \beta) \models [\![\texttt{r = res;}\ldots]\!]\varphi,$$

and this is property (thm1-goal') which we aimed to show. □

2.3 Dependency Contracts

Theorem 2 (Soundness of useDependencyContract). *Let* $\Gamma, \Delta \in 2^{Fma_\Sigma}$, $obs \in \mathcal{F} \cup \mathcal{P}$, $h^{new} = (f_1(f_2(\ldots(f_m(h^{base}, \ldots)))), \texttt{o}, \texttt{p}'_1, \ldots, \texttt{p}'_n) \in Trm_\Sigma$, $A \in \mathcal{T}$, $depct = (obs, \texttt{this}, (\texttt{p}_1, \ldots, \texttt{p}_n), pre, dep)$, $\texttt{hPre} \in \mathcal{PV}$, $mod = allLocs \mathbin{\dot{\setminus}} dep$, *reachableState*, *frame*, *noDeallocs* $\in Fma_\Sigma$, $w \in Upd_\Sigma$, *guard*, *equal* $\in Fma_\Sigma$ *all be as in Def. 7 of [2]. If*

$$\models \Gamma,\ guard \rightarrow equal\ \Rightarrow\ \Delta \tag{25}$$

and if for all types $B \sqsubseteq A$ *we have*

$$\models CorrectDependencyContract(depct, B), \tag{26}$$

then the following holds:

$$\models \Gamma \Rightarrow \Delta.$$

Proof. Let (25) and (26) hold, and let $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$ be a Kripke structure. Our goal is to show $(\mathcal{K}, s, \beta) \models \Gamma \Rightarrow \Delta$. We will do a proof by contradiction and assume that this does *not* hold, or in other words, that $(\mathcal{K}, s, \beta) \models \bigwedge(\Gamma \cup \neg\Delta)$ holds. This and (25) imply $(\mathcal{K}, s, \beta) \models \neg(guard \rightarrow equal)$, which means that $(\mathcal{K}, s, \beta) \models guard \wedge \neg equal$. If we insert the definitions of *guard* and *equal*, and distribute the update w over the conjuncts of *guard*, then this reads as

$$\begin{aligned}
&(\mathcal{K}, s, \beta) \models \{w\}\{\mathtt{heap} := h^{base}\}(pre \wedge reachableState)\\
&(\mathcal{K}, s, \beta) \models \{w\}\{\mathtt{hPre} := h^{base} \parallel \mathtt{heap} := h^{new}\}(frame \wedge noDeallocs)\\
&(\mathcal{K}, s, \beta) \models \neg\big(obs(h^{new}, \mathtt{o}, \mathtt{p}'_1, \ldots, \mathtt{p}'_n) \equiv obs(h^{base}, \mathtt{o}, \mathtt{p}'_1, \ldots, \mathtt{p}'_n)\big) \qquad (27)
\end{aligned}$$

Let $s_1 = val_{\mathcal{K},s,\beta}(w)(s)$. Then the first two statements above become

$$\begin{aligned}
&(\mathcal{K}, s_1, \beta) \models \{\mathtt{heap} := h^{base}\}(pre \wedge reachableState)\\
&(\mathcal{K}, s_1, \beta) \models \{\mathtt{hPre} := h^{base} \parallel \mathtt{heap} := h^{new}\}(frame \wedge noDeallocs)
\end{aligned}$$

Let $s_1^{base} = val_{\mathcal{K},s,\beta}(\mathtt{heap} := h^{base})(s_1)$, $s_1^{new} = val_{\mathcal{K},s,\beta}(\mathtt{hPre} := h^{base} \parallel \mathtt{heap} := h^{new})(s_1)$. Then the statements above turn into

$$(\mathcal{K}, s_1^{base}, \beta) \models pre \wedge reachableState \tag{28}$$

$$(\mathcal{K}, s_1^{new}, \beta) \models frame \wedge noDeallocs \tag{29}$$

As $\mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n$ do not occur in (27), and as s and s_1 are otherwise identical, we get by Prop. 1 that

$$(\mathcal{K}, s_1, \beta) \models \neg\big(obs(h^{new}, \mathtt{o}, \mathtt{p}'_1, \ldots, \mathtt{p}'_n) \equiv obs(h^{base}, \mathtt{o}, \mathtt{p}'_1, \ldots, \mathtt{p}'_n)\big),$$

which because of the definition of s_1 implies that

$$(\mathcal{K}, s_1, \beta) \models \neg\big(obs(h^{new}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \equiv obs(h^{base}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n)\big). \tag{30}$$

Lemma 1 and (29) tell us that

$$(\mathcal{K}, s_1^{new}, \beta) \models \mathtt{heap} \doteq anon(\mathtt{hPre}, \{\mathtt{heap} := \mathtt{hPre}\}mod, \mathtt{heap}),$$

which because of the definition of s_1^{new} is the same as

$$(\mathcal{K}, s_1, \beta) \models h^{new} \doteq anon(h^{base}, \{\mathtt{heap} := h^{base}\}mod, h^{new}). \tag{31}$$

Let $C = \delta(s_1^{base}(\mathtt{this}))$. This means that

$$(\mathcal{K}, s_1^{base}, \beta) \models exactInstance_C(\mathtt{this}). \tag{32}$$

Let $\mathcal{K}' = (\mathcal{D}, \delta, I', \mathcal{S}, \rho)$ be a Kripke structure identical to $\mathcal{K}$, except that $I'(h) = val_{\mathcal{K},s_1,\beta}(h^{new})$. Since $\alpha(\mathtt{this}) = A$, we have $C \sqsubseteq A$. Instantiating (26) with C, $\mathcal{K}'$ and s_1^{base} yields

$$\begin{aligned}(\mathcal{K}', s_1^{base}, \beta) \models\ & pre \wedge reachableState \wedge exactInstance_C(\mathtt{this}) \\ & \rightarrow obs(\mathtt{heap}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \\ & \quad \equiv \{\mathtt{heap} := anon(\mathtt{heap}, mod, h)\} \\ & \qquad obs(\mathtt{heap}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n).\end{aligned}$$

As h does not occur in (28) or (32), we have $(\mathcal{K}', s_1^{base}, \beta) \models pre \wedge reachableState \wedge exactInstance_C(\mathtt{this})$ by Prop. 2, which we can combine with the statement above to get

$$\begin{aligned}(\mathcal{K}', s_1^{base}, \beta) \models\ & obs(\mathtt{heap}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \\ & \equiv \{\mathtt{heap} := anon(\mathtt{heap}, mod, h)\} obs(\mathtt{heap}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n).\end{aligned}$$

Applying the update yields

$$\begin{aligned}(\mathcal{K}', s_1^{base}, \beta) \models\ & obs(\mathtt{heap}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \\ & \equiv obs(anon(\mathtt{heap}, mod, h), \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n).\end{aligned}$$

Because of the definition of s_1^{base}, this is the same as

$$\begin{aligned}(\mathcal{K}', s_1, \beta) \models\ & obs(h^{base}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \\ & \equiv obs(anon(h^{base}, \{\mathtt{heap} := h^{base}\}mod, h), \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n).\end{aligned}$$

By definition of $\mathcal{K}'$, we have $I'(h) = val_{\mathcal{K},s_1,\beta}(h^{new})$. As h does not occur in h^{new}, and as $\mathcal{K}$ and $\mathcal{K}'$ are otherwise identical, Prop. 2 guarantees that $val_{\mathcal{K},s_1,\beta}(h^{new}) = val_{\mathcal{K}',s_1,\beta}(h^{new})$. Thus, we have $I'(h) = val_{\mathcal{K}',s_1,\beta}(h^{new})$, and can thus write the statement above as

$$\begin{aligned}(\mathcal{K}', s_1, \beta) \models\ & obs(h^{base}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \\ & \equiv obs(anon(h^{base}, \{\mathtt{heap} := h^{base}\}mod, h^{new}), \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n).\end{aligned}$$

As the function symbol h does not occur in the above formula, ans as $\mathcal{K}$ and $\mathcal{K}'$ are otherwise identical, Prop. 2 tells us that

$$\begin{aligned}(\mathcal{K}, s_1, \beta) \models\ & obs(h^{base}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \\ & \equiv obs(anon(h^{base}, \{\mathtt{heap} := h^{base}\}mod, h^{new}), \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n).\end{aligned}$$

We can combine this with (31) to get

$$(\mathcal{K}, s_1, \beta) \models obs(h^{base}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n) \equiv obs(h^{new}, \mathtt{this}, \mathtt{p}_1, \ldots, \mathtt{p}_n),$$

which contradicts (30). □

References

1. J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification, Second Edition*. Addison-Wesley, 2000.
2. P. H. Schmitt, M. Ulbrich, and B. Weiß. Dynamic frames in Java dynamic logic. In B. Beckert and C. Marché, editors, *Proceedings, International Conference on Formal Verification of Object-Oriented Software (FoVeOOS 2010)*, LNCS. Springer, 2010. To appear.